# Homework 2

# Studente : Miro Filomeno Davide 256870

## 0)Code and data analysis

The Caltech dataset provides 8095 different images divided in 101 This dataset is very diversified and the images have different resolution. The providen code first define preprocessing trasformation. Images are first resized to reach the resolution 224x224 that is one accepted by AlexNet,then trasform to pytorch tensors and finally there are normalization.After preprocessing dataset are divided in 2 split, train*dataset adn test*dataset in proportion 5:1 without following the suggested split on test.txt and train.txt. Data are putted in the model as batch of 256 elements,so as optimization algorithm is choosen SGD with momentum, a learning rate equal to $\epsilon$ = 0.001 and a momentum paramiter $\boldsymbol{\alpha}$ = 0.9.As regularization strategy is implemented the learning rate decay at epoch 20 with a gamma = 0.01.A first run of code give an accuracy on test set equal to 0.09

## 1) Data preparation

In the dataset there are 2 problems:

- The split strategy providen by test.txt and train.txt is not implemented
- In the dataset there are the **BACKGROUND_Google** folder that contains some pictures that are useless for the analysis

To solve this 2 problem is implemented a class Caltech that is responsible to access to images associated to a label This is the code:

```
def pil_loader(path):
    # open path as file to avoid ResourceWarning (https://github.com/python-pillow/P
    with open(path, 'rb') as f:
        img = Image.open(f)
        return img.convert('RGB')
class Caltech():
    def __init__(self, root, split, transform=None):

        self.root = root
        self.transform= transform

        self.split = split # This defines the split you are going to use
                           # (split files are called 'train.txt' and 'test.txt')
        target_lines = [l.split("/")[0] for l in self.split]
        targets = []
        for i in range(len(target_lines)):
            if i == 0 or target_lines[i - 1] != target_lines[i]:
                targets.append(target_lines[i])
        self.map_target = {targets[i] : i  for i in range(len(targets))}

    def __getitem__(self, index):
        '''
        __getitem__ should access an element through its index
        Args:
            index (int): Index

        Returns:
            tuple: (sample, target) where target is class_index of the target class.

        image, label = pil_loader(self.root+"/"+self.split[index]),self.map_target[s
        if self.transform is not None:
            image = self.transform(image)

        return image, label

    def __len__(self):
        '''
        The __len__ method returns the length of the dataset
        It is mandatory, as this is used by several other components
        length = len(self.split)
        return length
```

The class receive in the costructor 3 paramiter:

- **root** is the path of folder in which there are class folders.It's the product of
- **split** is a list that contains all relative paths of the images
- **trasform** trasformation on data

This is the code which use the Caltech class.

```
DATA_DIR = 'Homework2-Caltech101/101_ObjectCategories'
f1 = open('Homework2-Caltech101/train.txt','r')
f2 = open('Homework2-Caltech101/test.txt','r')

train_split = f1.readlines()
test_split = f2.readlines()

train_split = [l.replace('\n', '') for l in train_split if l.startswith("BACKGROUN
D_Google") == False]
test_split = [l.replace('\n', '') for l in test_split if l.startswith("BACKGROUND_
Google") == False]


# Prepare Pytorch train/test Datasets
train_dataset = Caltech(DATA_DIR,train_split,train_transform)
test_dataset = Caltech(DATA_DIR,test_split,eval_transform)
```

The reading on train.txt and text.txt produces two lists train*split and test*split and in which of them are filtered images from **BACKGROUND_Google** folder.

## 2A) Split train set in validation set and train set

To evaluate the model at each epoch and select the best set of paramiters train test is splitted in two subsets:

```
validation_split = []
train2_split = []
j = 0
for i in range(len(train_split)):
  j = (j + 1) % 2
  if i != 0 and train_split[i].split("/")[0] != train_split[i - 1].split("/")[0]:
    j = 0
  if j > 0:
    train2_split.append(train_split[i])
  else:
    validation_split.append(train_split[i])
```

Given the list train_split that comes from train.txt as shown before it is splitted in two sublist as shown in the code above.

## 2B) Implement model selection with validation

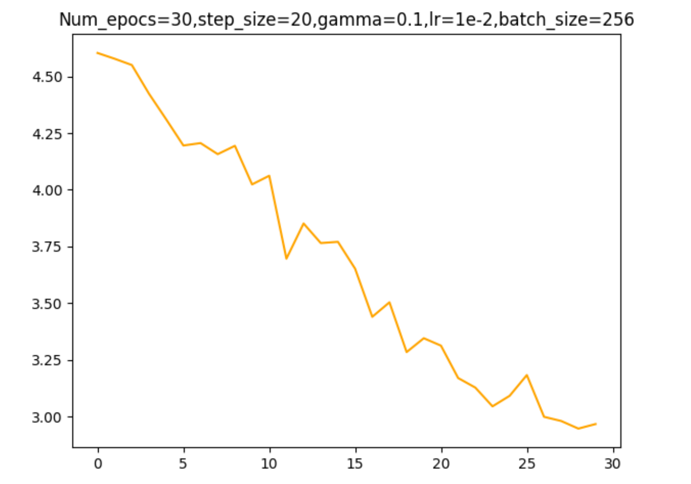
Model selection is implemented taken a test phase over the validation_set in the train phase at the end of each epoch.

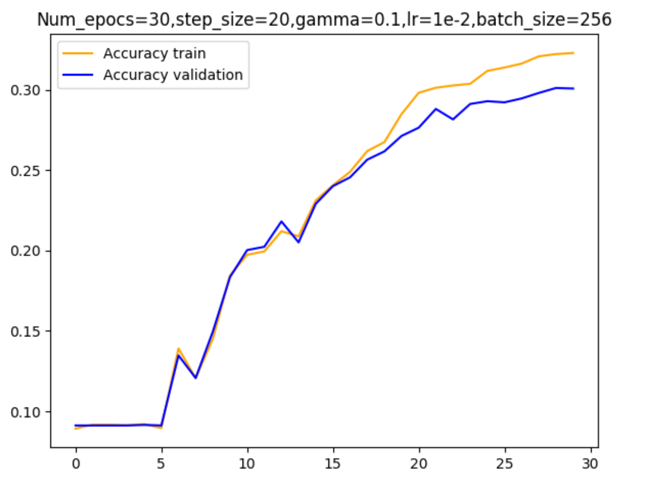
```
  net.train(False)
  running_corrects = 0
  for images, labels in tqdm(validation_dataloader):
    images = images.to(DEVICE)
    labels = labels.to(DEVICE)
    outputs = net(images)
    _, preds = torch.max(outputs.data, 1)
    running_corrects += torch.sum(preds == labels.data).data.item()
  accuracy = running_corrects / float(len(validation_dataset))
  if(maxAccuracy < accuracy):
    maxAccuracy = accuracy
    bestNet = net
```

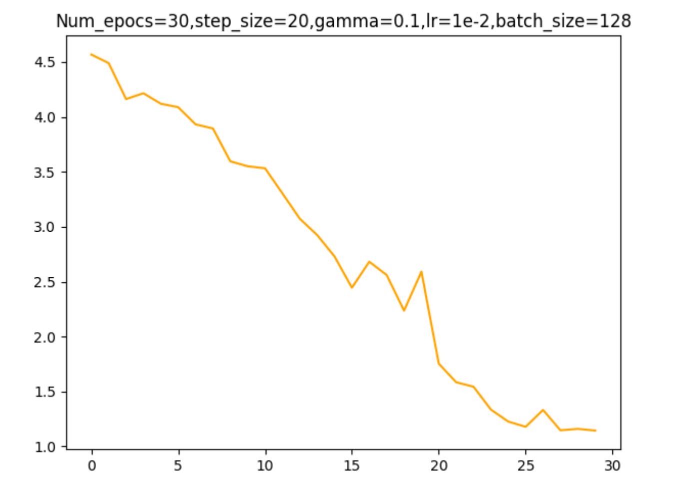
The final model is one that in a certain epoch give the best accuracy on validation set. Running the algorithm with this update produce a model with a best accuracy on validation set equal to **0.0912** and an accuracy on test set of **0.0919**.

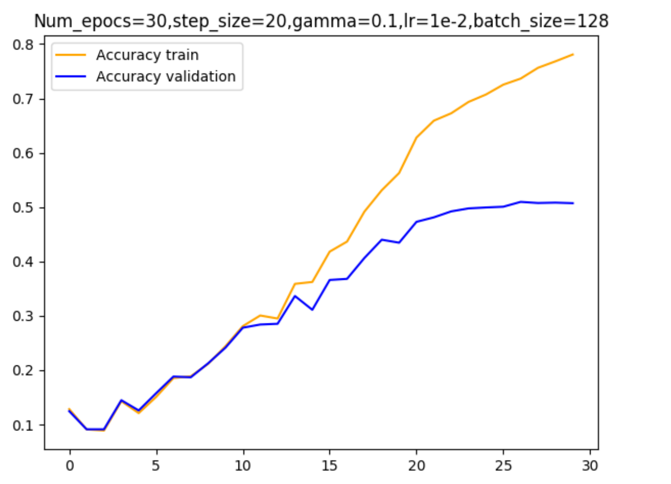
## 2C) Experiment with at least 2 set of hyperparamiters

As you can see that the performance of model are very bad and the accuracy increase very slow.To learning rate is up to 0.1.This setting make diverge the model,the loss becomes **NaN** in few epochs.A median value to give at learning rate is 0.01. The generated model over this new set of hyperparamiter give an accuracy of **0.301** and **0.302** over validation and test set respectively.Another trick that can be used to get a more accurate model is to increase batch size to **128**.The training process so setted produce a model with accuracy of **0.509** for the validation set and **0.503** for test set.The graphs of the loss and accuracies of two configurations of hyperparamiters are shown below.





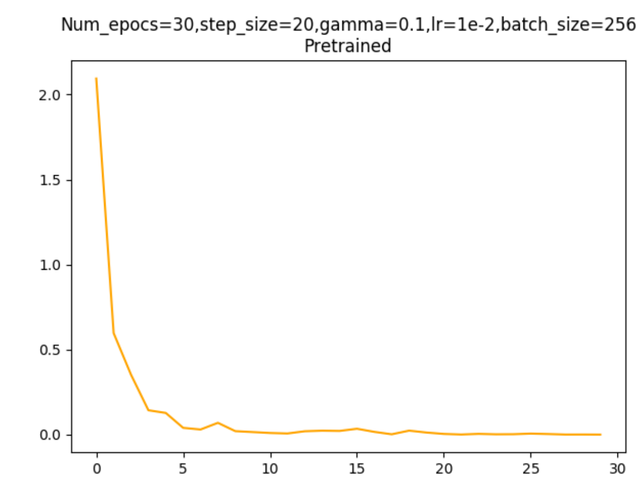


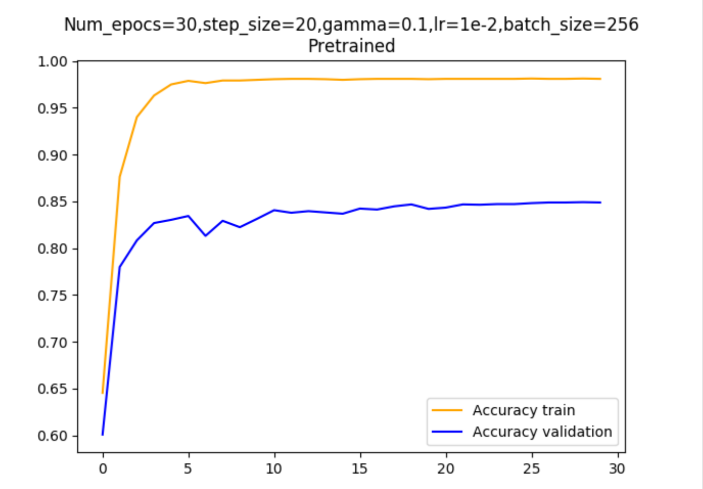


Decreasing the batch_size improve performance of model but increase the gap between the training set and validation set.

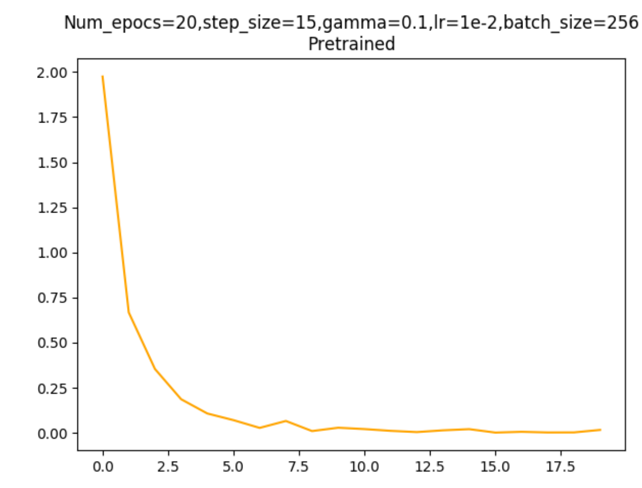
## 3) Transfer Learning

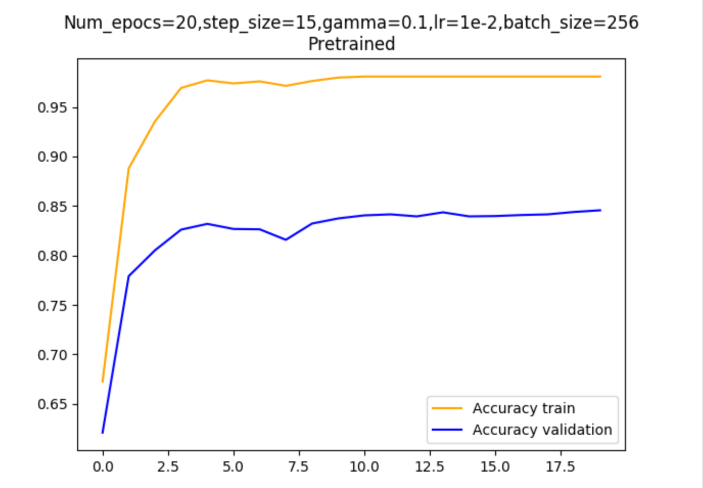
It's clear that the model doesn't perform well and this is due to lack of data for so complex problem.To get a better model it's possible to learn a pre-learned model over ImageNet dataset.First import the model setting to True the paramiter **pretrained** of alexnet class costructor.Then adapt the available dataset to ImageNet one changing means and stds of each of the 3 channels of the images to the values of ImageNet dataset **means = [0.485, 0.456, 0.406] and stds = [0.229, 0.224, 0.225]**. The hyperparamiters of the first run are: **num_epochs = 30**,**step_size = 20**,**gammma = 0.1**,**learning_rate = 0.01** and **batch_size =256**. The result is a network with a peak of vealidation accuracy equal to **0.849** and an accuracy on test set equal to **0.854**.The trend of loss and accuracy is shown in the following graphs:



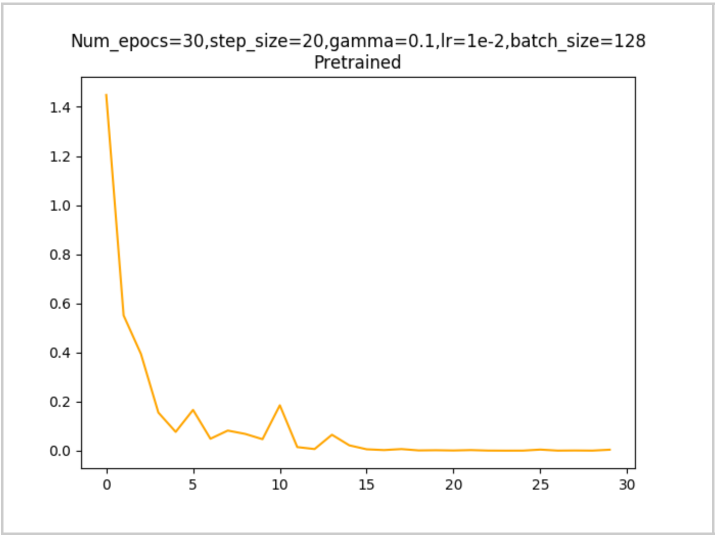


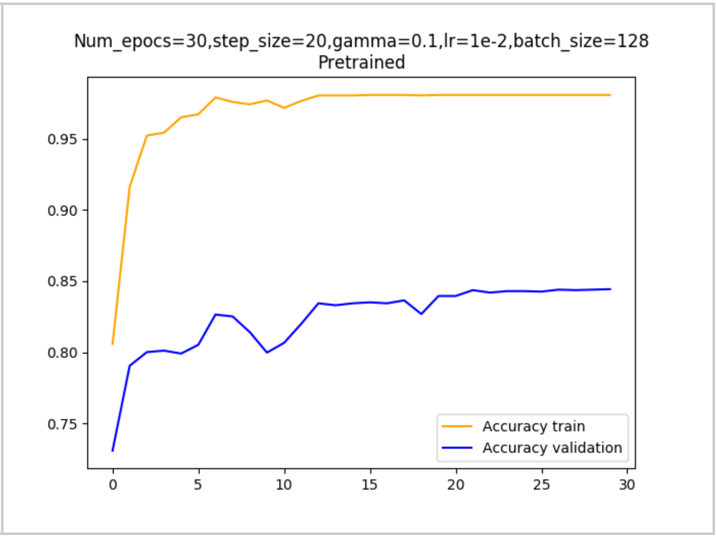
As you can see the performance of model in terms of accuracy improves drammatically but remains the problem of overfitting.To try to solve this problem I followed 2 way.The first one is to reduce the number of epochs because as you can the model in training phase faster reach higher values of accuracy, so i tried to put **num_epochs = 20**, **step_size=15**, **gammma = 0.1**,**learning_rate = 0.01** and **batch_size =256**.The algorithm provides a model of max validation accuracy = **0.844** and test accuracy = **0.845**.The graphs:





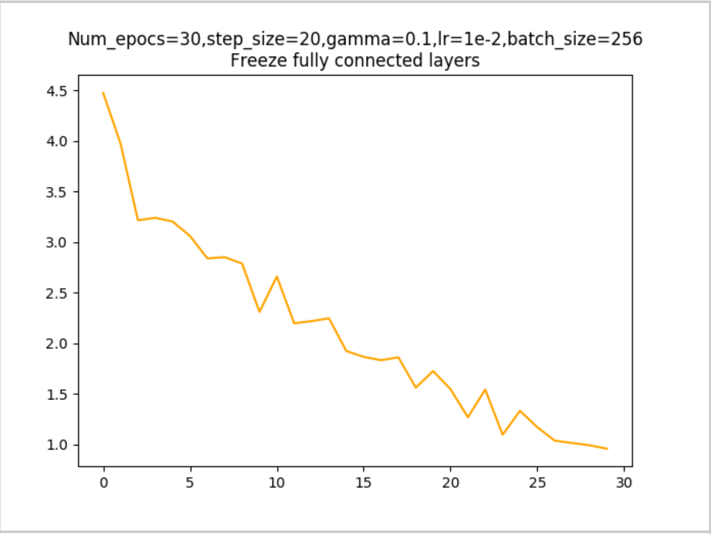
Another strategy is to reduce batch size = 128 in order to have a more accurate learning process.The result is a model with best validation accuracy = **0.844** and test accuracy = **0.843.**These are the graphs:

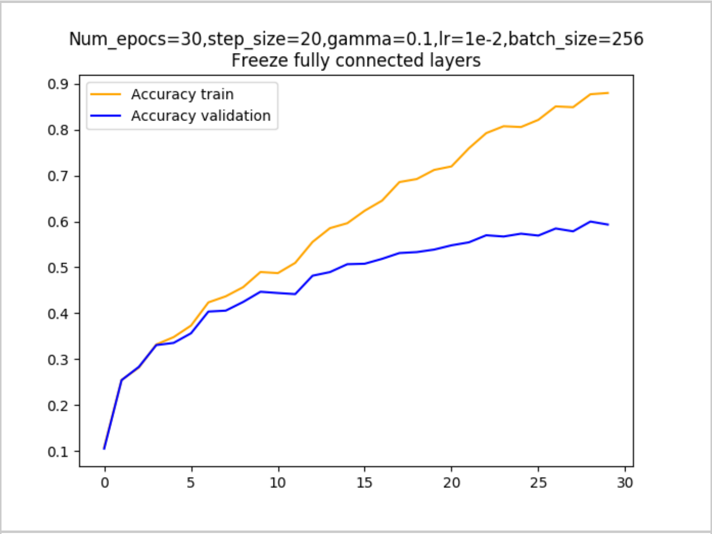


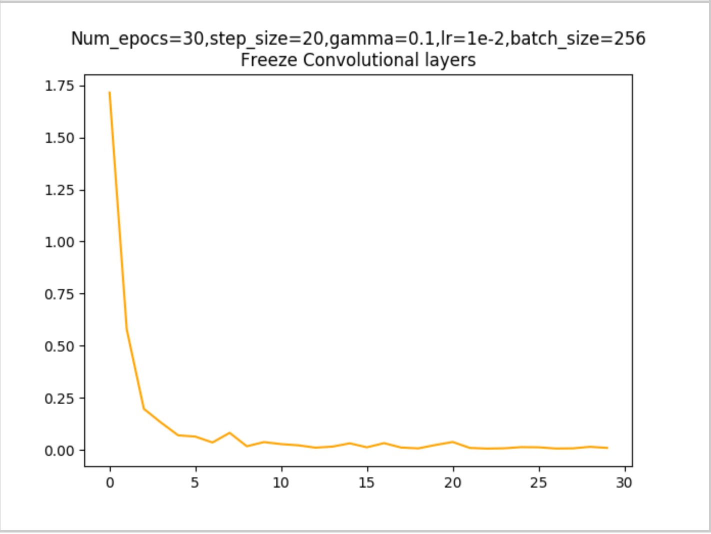


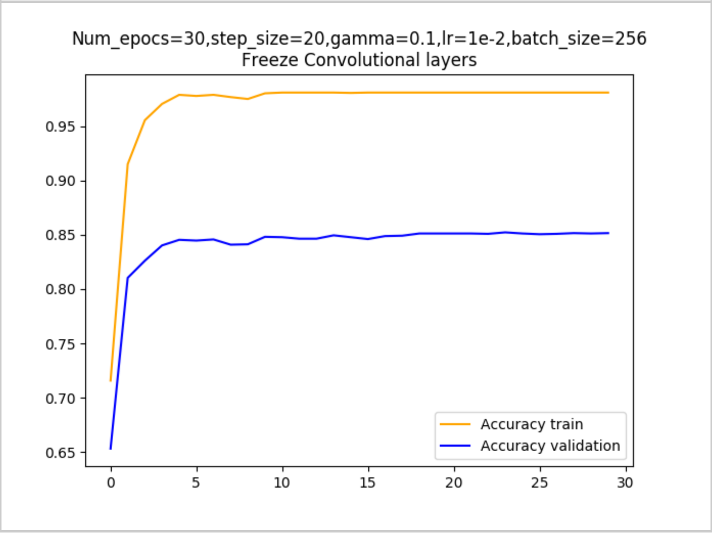
A different learning approach is to freeze some parts of pretrained network.First I freezed the fully connected layer and I start learning using the hyperparamiters tuned before (**num_epochs = 30**,**step_size = 20**,**gammma = 0.1**,**learning_rate = 0.01**),the result is disastrous,the performace decrease to **0.599** for best validation and **0.57** for test set.This happens because the convolutional layers are already learned to extract features,in this case we want to learn model to classify the new classes so we must learn only fully connected layers.We do that freezing the convolutional layers and the resulting model obtain a best validation accuracy of **0.854** and a test accuracy of **0.857** Next are shown the code used for freezing the model and the graphs of loss and accuracy of both two cases.

```
parameters_to_optimize = net.classifier.parameters()#freeze conv layer
parameters_to_optimize = net.features.parameters()#freeze fully layer
```



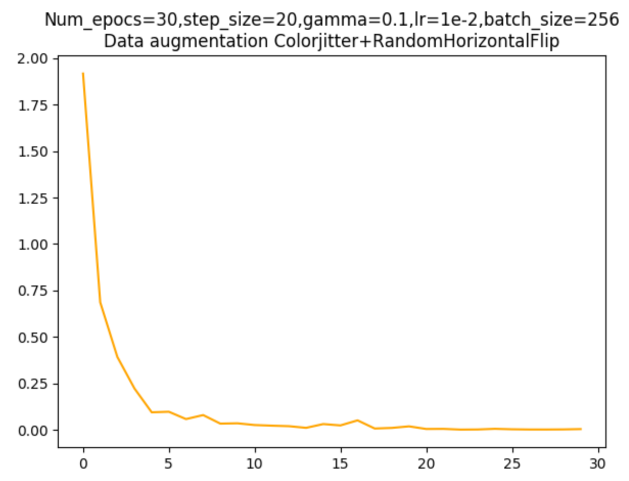


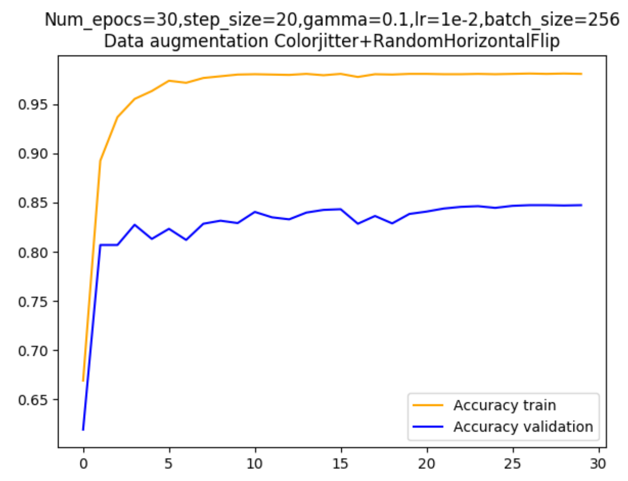


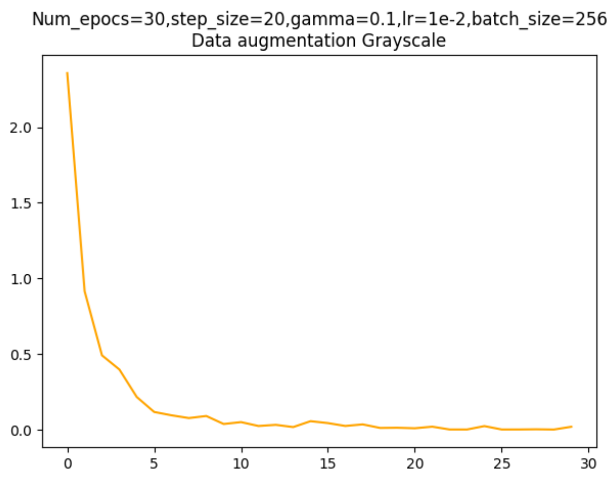


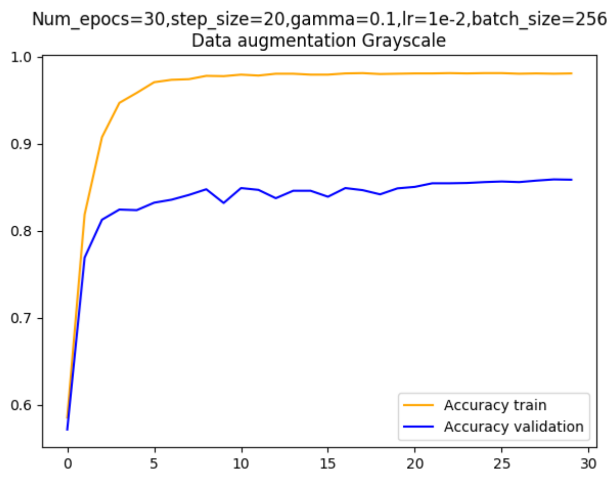
## 4)Data augmentation

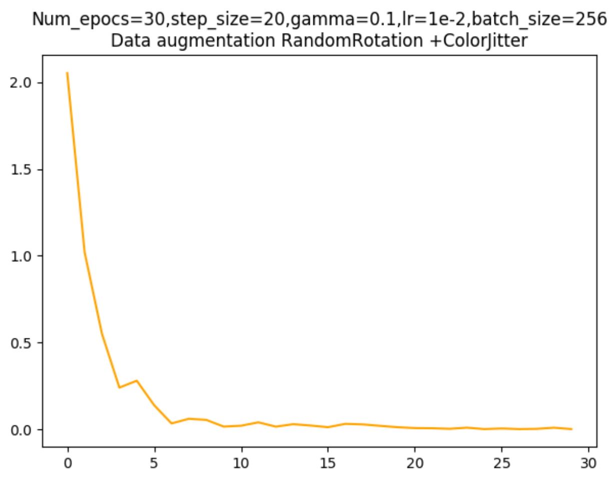
In order to have more data to use in training phase it's possible to perform some random transformations over the training data to have new samples.This strategy is called data augmentation and it's obtained adding new trasformations before preprocess ones.In this homework are used **ColorJitter,RandomGrayscale,RandomHorizontalFlip and RandomRotation**. The first model is obtained using as data augmentation transformation **ColorJitter** and **RandomHorizontalFlip** and as hyperparamiters **num_epochs = 30**,**step_size = 20**,**gammma = 0.1**,**learning_rate = 0.01** and **batch_size =256**.The model obtain a best validation accuracy of **0.849** and a test accuracy of **0.85**.With the same hyperparamiters are tried other 2 set of random transformation: a first execution with **RandomGrayscale** reachs accuracies of **0.859** and **0.853** for the best validation score and test set respectively; a second execution with **ColorJitter** and **RandomRotation** that get **0.849** and **0.850** The graphs of the three execution are shown below:

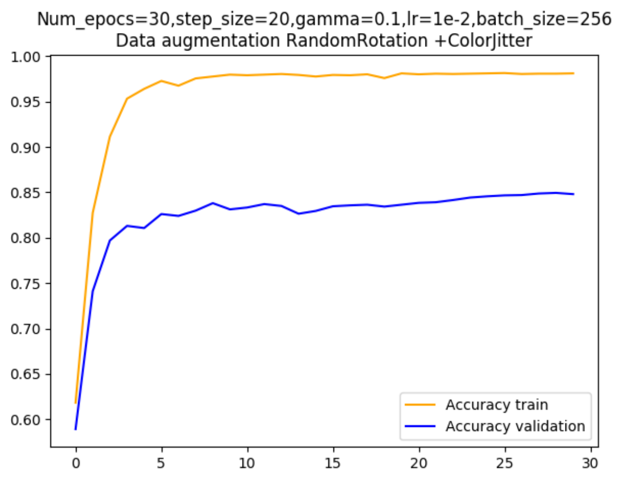






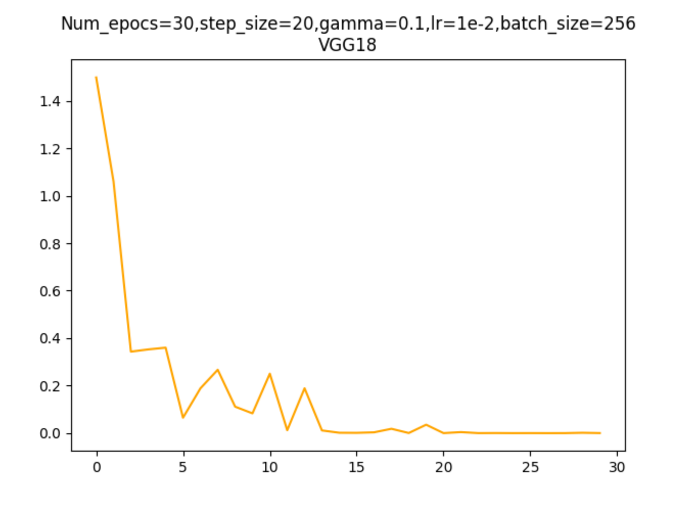


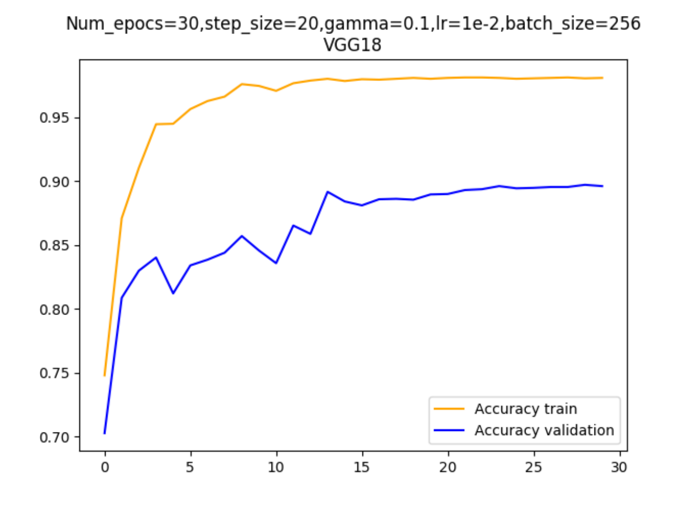


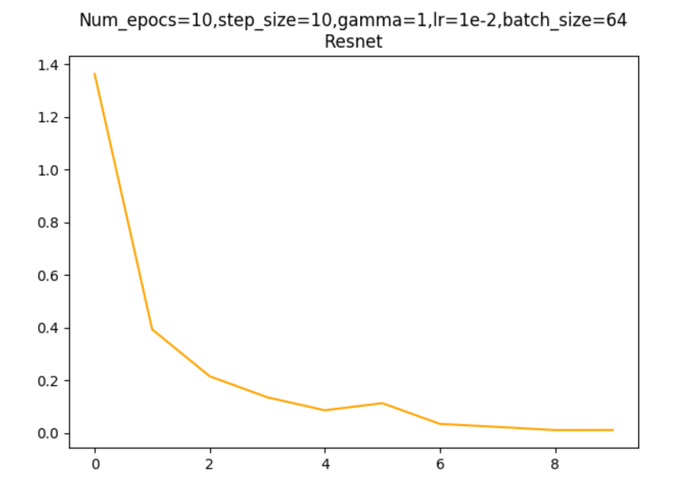


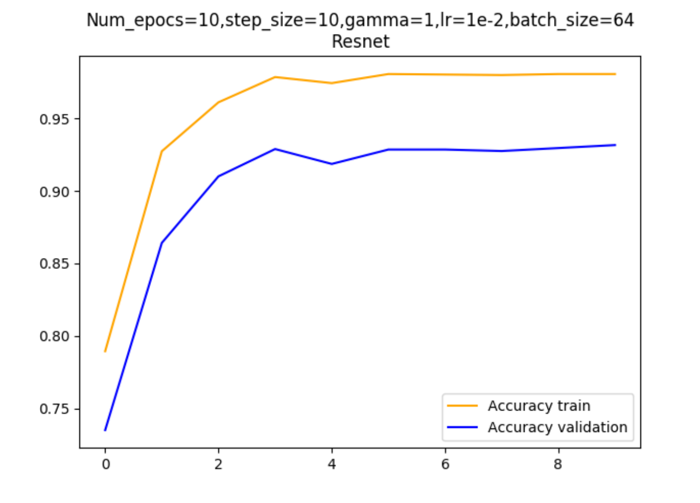
## 5)(Extra) Beyond AlexNet

I have also tried to learn different CNNs from AlexNet such that VGG11 and resnet.For **VGG11** I tried to reuse the same hyperparamiters tuned with alexnet (**num_epochs = 30**,**step_size = 20**,**gammma = 0.1**,**learning_rate = 0.01** and **batch_size =256**).The give model provide best validation accuracy = **0.897** and test accuracy = **0.899.** For resnet considering that is a very deep CNN I reduced batch_size = 64 and the number of epochs to 10.The resulting model obtain best validation accuracy = **0.931** and test accuracy of **0.937**.The graphs of the model are shown below:


Num_epocs=30,step_size=20,gamma=0.1,lr=1e-2,batch_size=256
VGG18


Num_epocs=30,step_size=20,gamma=0.1,lr=1e-2,batch_size=256
VGG18
Accuracy train
Accuracy validation


Num_epocs=10,step_size=10,gamma=1,lr=1e-2,batch_size=64
Resnet


Num_epocs=10,step_size=10,gamma=1,lr=1e-2,batch_size=64
Resnet
Accuracy train
Accuracy validation